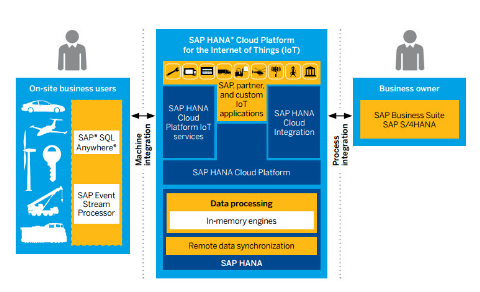
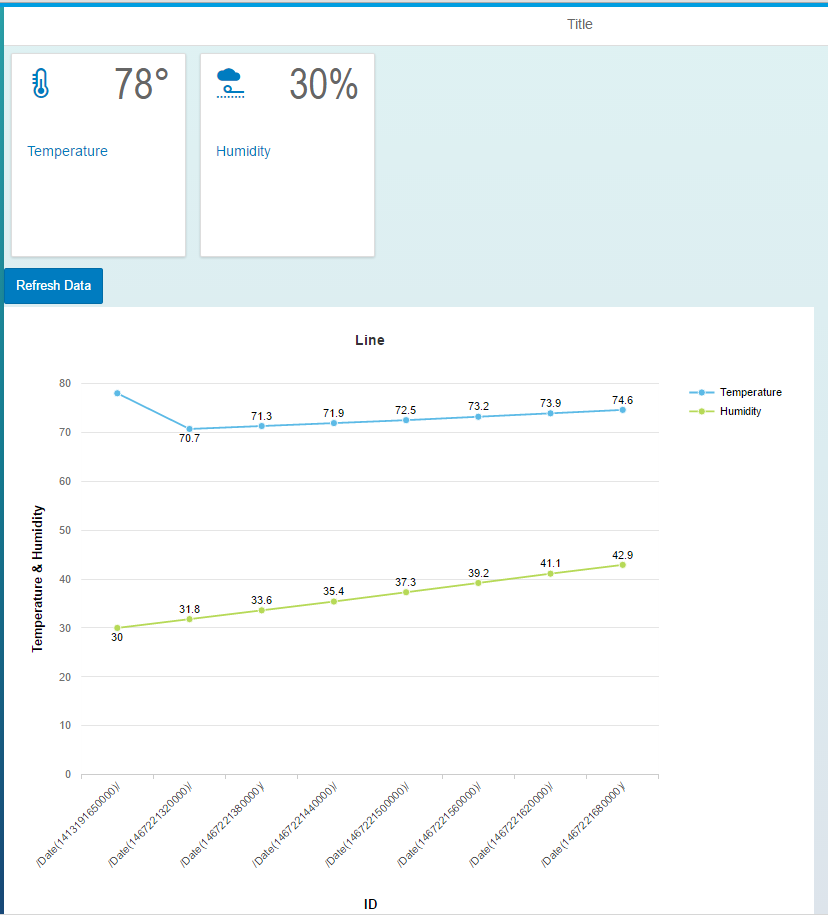
IoT1C03 – Create a Dashboard - HCP

|  |  |
| --- | --- |
| **Product**  HANA Cloud Platform IoT Services  **Level**  Undergraduate/Graduate  Beginner  **Focus**  HANA Cloud Platform  **Author** Ross Hightower | MOTIVATION  In this case you will create a simple dashboard for the climate message type data.  **PREREQUISITES**  IoT1C01 – Setup IoT Services - HCP |



# Create a Dashboard

In this case you will create a small UI5 application to display the data transmitted by the Tessel. The application looks like the image below. The two tiles at the top show the most recent temperature and humidity and the chart shows the last 10 values. The application refreshes every 3 seconds but the refresh can be toggled off using the button.

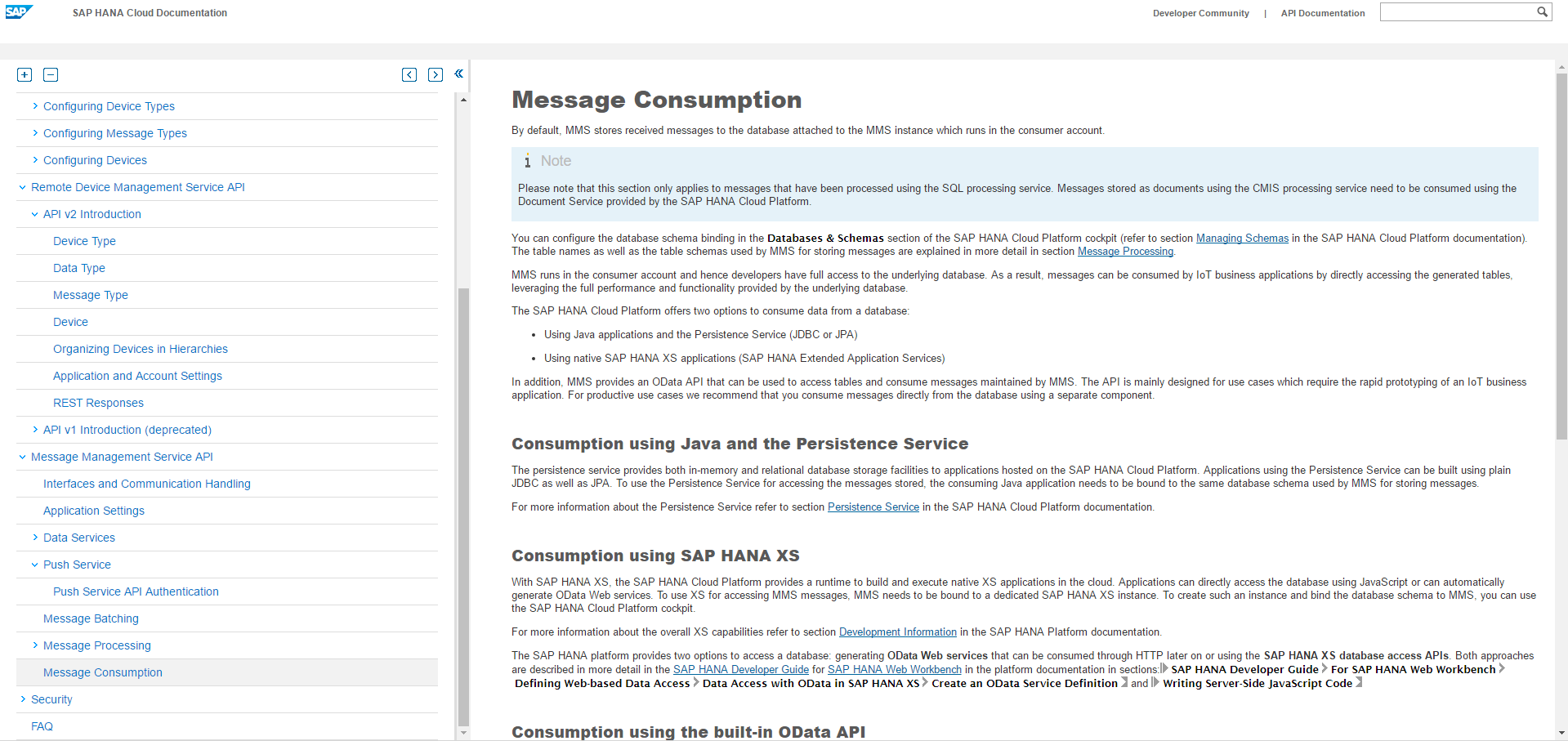


## Accessing the Data

Since the messages are stored in a HANA database table you can use all the facilities available in HANA to create services and applications that consume those services. However, the MMS includes some basic OData services that you can use if you don’t require anything custom. To see documentation on the services, navigate to:

<https://help.hana.ondemand.com/iot>

Scroll down **Message Consumption** documentation.



The details are in the section titled Consumption using the built-in OData API.



For example, to retrieve the service document use this URI:

https://iotmms<your id>trial.hanatrial.ondemand.com/com.sap.iotservices.mms/v1/api/http/app.svc

Paste the URI into a browser and you will get the service document (if prompted, enter your HCP user id and password):



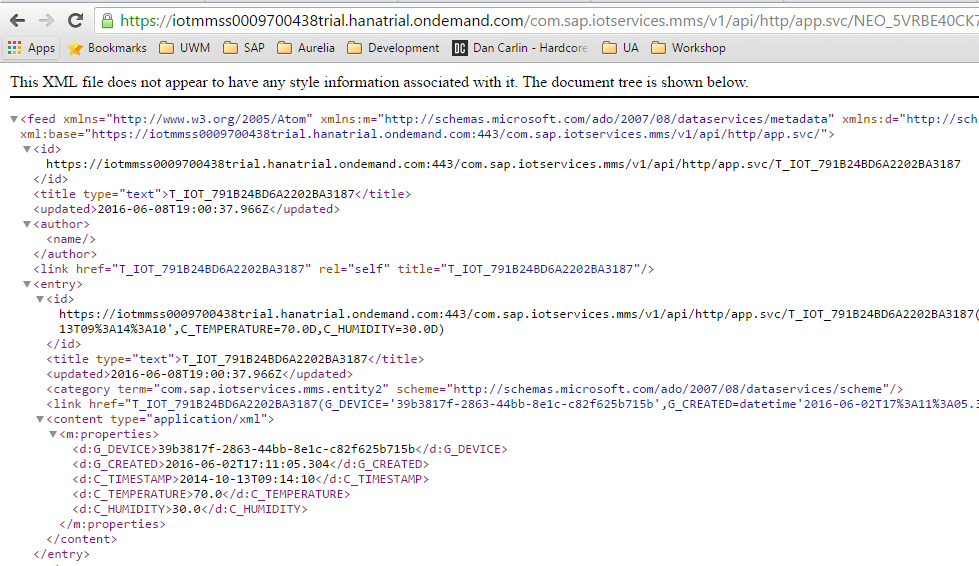
There is a service for each table. Tables with messages from devices will look like the one below. The part starting with NEO… is the database schema created for the IoT services and the portion after the dot is the table name. Note the message type ID is included in the table name.



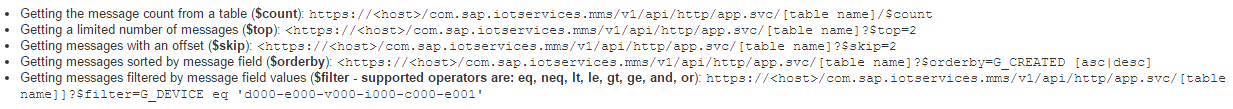
To see the data in the table, add a slash and then the href value (/<schema>.<table>) to the service document URI.

https://iotmms<your id>trial.hanatrial.ondemand.com/com.sap.iotservices.mms/v1/api/http/app.svc/NEO\_5VRBE40CK71NRAETYPE3ZRSFGX.T\_IOT\_791B24D6A2202BA3187

The result is the message data in XML format (the results you see will depend on the browser and whether you have installed an extension to format the code. You can also use Postman to see the results):

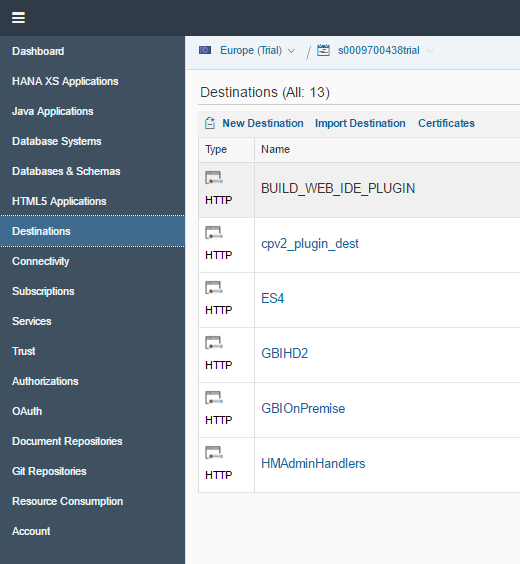


The documentation shows other options you can use with OData services:

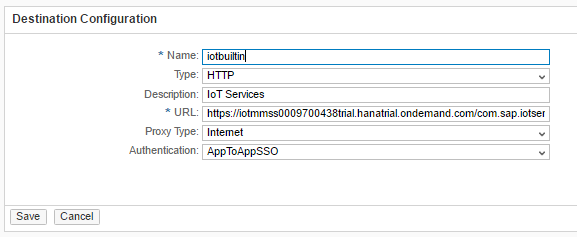


## Create a Destination

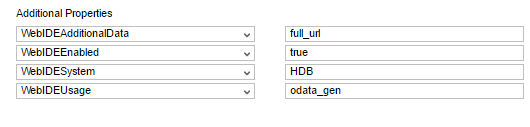
To access the data from within HCP we have to create a Destination in the HCP Cockpit. Logon to the HCP Cockpit and navigate to Destinations.



Click **New Destination**. Enter the data shown below (The URL is the service document URI):



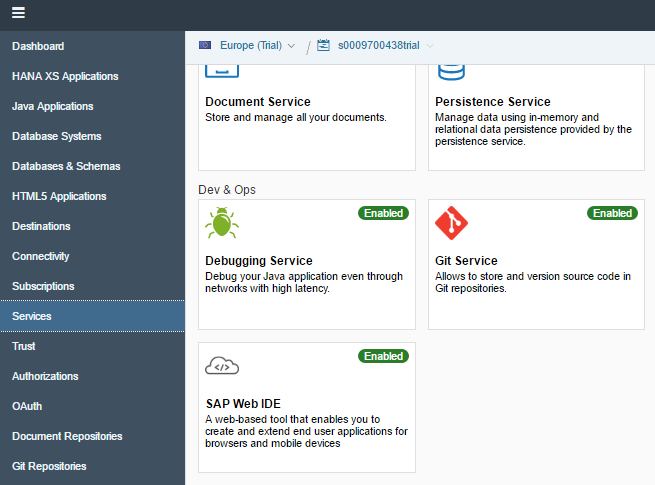
Next, use the **New Property** button to add the following properties:



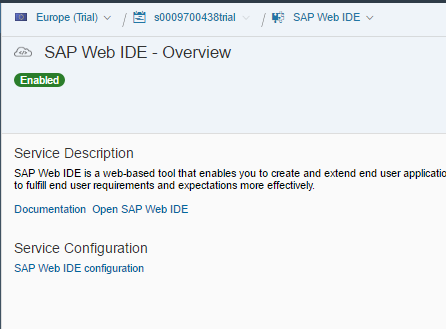
Click **Save**. Use the **Check Connection** button to test the destination.

## Create the Application

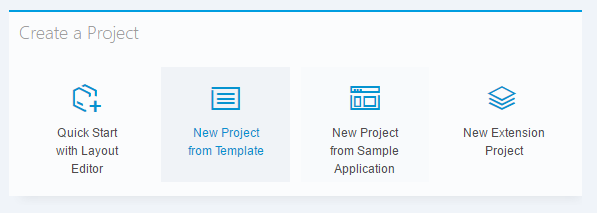
Open the WebIDE. To open the WebIDE, navigate to the Services page in the HCP Cockpit.



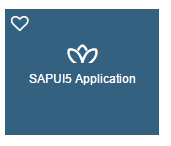
Locate the SAP Web IDE tile and click it. If it isn’t enabled, click Enable. Click **Open SAP Web IDE.** You can bookmark this URL to return the WebIDE directly.



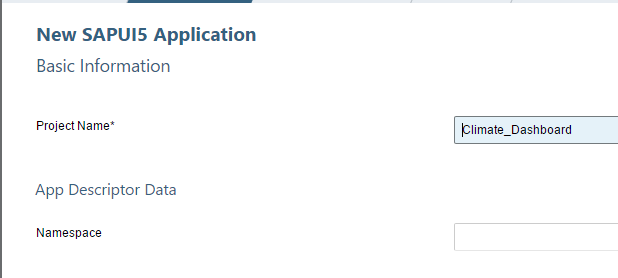
To create a new project, click **New Project from Template.**



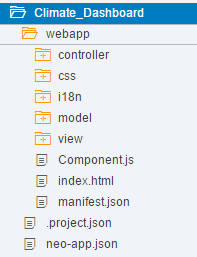
Choose the SAPUI5 Application template and click **Next**.



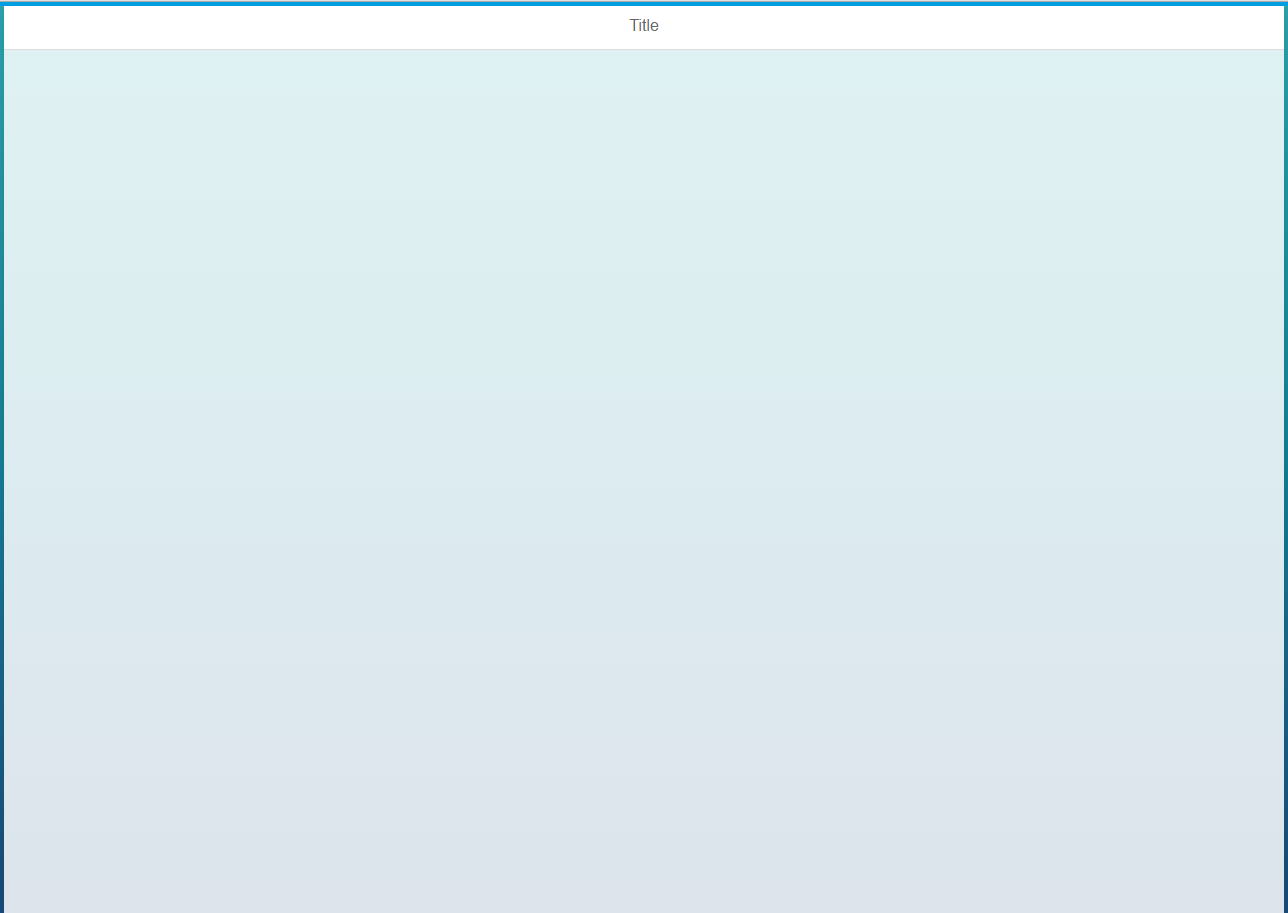
Enter a name and click **Finish.**



The project is created.



You can run the application now by right-clicking the project and selecting **Run→Run As→Web Application.** I doesn’t have any content yet so we will need to add our code.



### neo-app.json

The first thing to do is to add a reference to the destination you created to the **neo-app.json** file so the application can find it. The file already contains references to SAPUI5 libraries. Copy the code below and paste it into the file. Use the following image as a reference as to where to paste it.

|  |
| --- |
| ,  {  "path": "/destinations/iotbuiltin",  "target": {  "type": "destination",  "name": "iotbuiltin"  },  "description": "IoT Services"  } |

Listing 1



### View1.view.xml

Open the View1.view.xml file and replace the code in it with the code shown below.

|  |
| --- |
| <mvc:View controllerName="Climate\_Dashboard.controller.View1" xmlns:html="http://www.w3.org/1999/xhtml" xmlns:mvc="sap.ui.core.mvc"  xmlns="sap.m" xmlns:viz="sap.viz.ui5.controls">  <App>  <pages>  <Page title="{i18n>title}">  <content>  <HBox>  <StandardTile id="tempTile" icon="sap-icon://temperature" title="Temperature" number="{/d/results/0/C\_TEMPERATURE}"/>  <StandardTile id="humTile" icon="sap-icon://weather-proofing" title="Humidity" number="{/d/results/0/C\_HUMIDITY}"/>  </HBox>  </content>  </Page>  </pages>  </App>  </mvc:View> |

Listing 2

This code implements the two tiles. The number property of the tiles are bound to the C\_TEMPERATURE and C\_HUMIDITY properties of the first record ( /0/ ) in the data model. We will instantiate the model in the controller.

### View1.controller.js

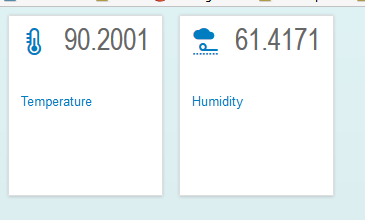
Open the **View1.controller.js** file and replace the code in with the code shown below.

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller",  "sap/ui/model/json/JSONModel"  ], function(Controller, JSONModel) {  "use strict";  return Controller.extend("Climate\_Dashboard.controller.View1", {  onInit: function() {  this.oModel = new sap.ui.model.json.JSONModel();  this.oModel.loadData(  '/destinations/iotbuiltin/NEO\_CT4FMF3FDW02NYG4UIPXV81BX.T\_IOT\_791B24BD6A2202BA3187?$format=json&$top=10&orderby=C\_TIMESTAMP%20desc'  );    this.getView().setModel(this.oModel);  }  });  }); |

Listing 3

This code implements the onInit function which will execute when the view is instantiated. The code initializes a JSON model and provides the URI of your service. Replace the highlighted portion so that it matches your URI. The URI includes some parameters that limit the data returned to the top 10 records ($top=10), formats the data as JSON ($format=json) and sorts the data by C\_TIMESTAMP in descending order. The result is that only the last ten records is returned in reverse order.

Run or refresh the application if it is already open.



Next, we will clean up the number formatting.

### View1.view.xml

Change the code for the StandardTiles in the View1.view.xml file to look like the code shown below. The highlighted portions show the changes.

|  |
| --- |
| <StandardTile id="tempTile" icon="sap-icon://temperature" title="Temperature" number="{path: '/d/results/0/C\_TEMPERATURE', formatter: '.degrees'}"/>  <StandardTile id="humTile" icon="sap-icon://weather-proofing" title="Humidity" number="{path: '/d/results/0/C\_HUMIDITY', formatter: '.humidity'}"/> |

This code configures functions (degrees() and humidity()) which are called before the value is displayed so that we can format the values.

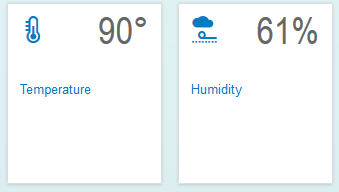
### View1.controller.js

Now add the highlighted portion below to the View1.controller.js file. Note the comma between the onInit() function and the degrees() function.

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller",  "sap/ui/model/json/JSONModel"  ], function(Controller, JSONModel) {  "use strict";  return Controller.extend("Climate\_Dashboard.controller.View1", {  onInit: function() {  this.oModel = new sap.ui.model.json.JSONModel();  this.oModel.loadData(  '/destinations/iotbuiltin/NEO\_CT4FMF3FDW02NYG4UIPXV81BX.T\_IOT\_791B24BD6A2202BA3187?$format=json&$top=10&orderby=C\_TIMESTAMP%20desc'  );    this.getView().setModel(this.oModel);  },    degrees: function(temp){  return Math.round(temp) + "\u00b0";  },    humidity: function(hum){  return Math.round(hum)+ "\u0025";;  }  });  }); |

Listing 4

These functions round the values to integers and add a degrees and percent sign. Now refresh or run the application again.



## Add a Chart

Next, we’ll add a chart. The JavaScript for the chart is a bit complex but if you look at it carefully you will see what it is doing.

### View1.view.xml

Make the following changes to the View1.view.xml file.

|  |
| --- |
| <mvc:View controllerName="Climate\_Dashboard.controller.View1" xmlns:html="http://www.w3.org/1999/xhtml" xmlns:mvc="sap.ui.core.mvc"  xmlns="sap.m" xmlns:viz="sap.viz.ui5.controls">  <App>  <pages>  <Page title="{i18n>title}">  <content>  <HBox>  <StandardTile id="tempTile" icon="sap-icon://temperature" title="Temperature" number="{path: '/d/results/0/C\_TEMPERATURE', formatter: '.degrees'}"/>  <StandardTile id="humTile" icon="sap-icon://weather-proofing" title="Humidity" number="{path: '/d/results/0/C\_HUMIDITY', formatter: '.humidity'}"/>  </HBox>  <ToggleButton id="refreshButtonID" text="Refresh Data" pressed="true" press="doIt"/>  <HBox>  <viz:Popover id="idPopOver"/>  <viz:VizFrame id="idVizFrameLine" height="700px" width="900px" vizType="line" uiConfig="{applicationSet:'fiori'}"></viz:VizFrame>  </HBox>  </content>  </Page>  </pages>  </App>  </mvc:View> |

Listing 5

### View1.controller.js

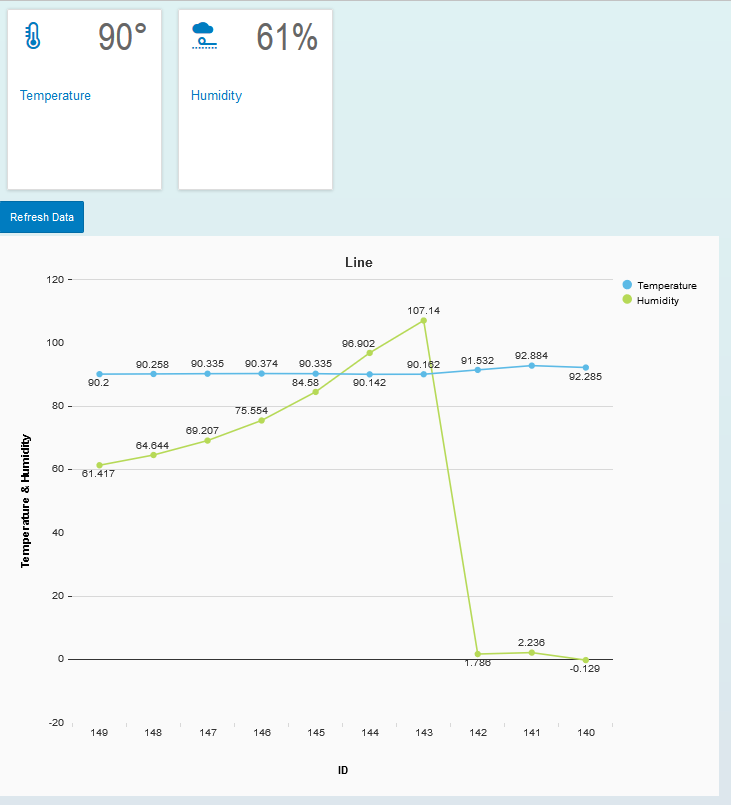
Now add the following code to the View1.controller.js file. The code is inserted into the onInit() function immediate after this line:

this.getView().setModel(this.oModel);

|  |
| --- |
| var oVizFrame = this.getView().byId("idVizFrameLine");  var oPopOver = this.getView().byId("idPopOver");  var oDataset = new sap.viz.ui5.data.FlattenedDataset({  dimensions: [{  name: 'ID',  value: "{C\_TIMESTAMP}"  }],  measures: [{  name: 'Temperature',  value: '{C\_TEMPERATURE}'  }, {  name: 'Humidity',  value: '{C\_HUMIDITY}'  }],  data: {  path: "/d/results"  }  });  oVizFrame.setVizProperties({  plotArea: {  isFixedDataPointSize: true,  categorySize: {  desktop: {  minValue: 100  }  },  dataLabel: {  visible: true  },  lineStyle: {  rules: [{  dataContext: [{  ID: "\*"  }],  properties: {  width: 6  }  }]  }  },  legend: {  title: {  visible: false  }  },  title: {  visible: true,  text: 'Line'  }  });  oVizFrame.setDataset(oDataset);  oVizFrame.setModel(this.oModel);  var feedPrimaryValues = new sap.viz.ui5.controls.common.feeds.FeedItem({  'uid': "primaryValues",  'type': "Measure",  'values': ["Temperature", "Humidity"]  }),  feedAxisLabels = new sap.viz.ui5.controls.common.feeds.FeedItem({  'uid': "axisLabels",  'type': "Dimension",  'values': ["ID"]  });  oVizFrame.addFeed(feedPrimaryValues);  oVizFrame.addFeed(feedAxisLabels);  oPopOver.connect(oVizFrame.getVizUid()); |

Listing 6

Refresh or run the application.



### Refresh the Data

The last change is to add the code that will cause the data to refresh automatically.

### View1.controller.js

Change the code in the View1.controller.js file as shown below.

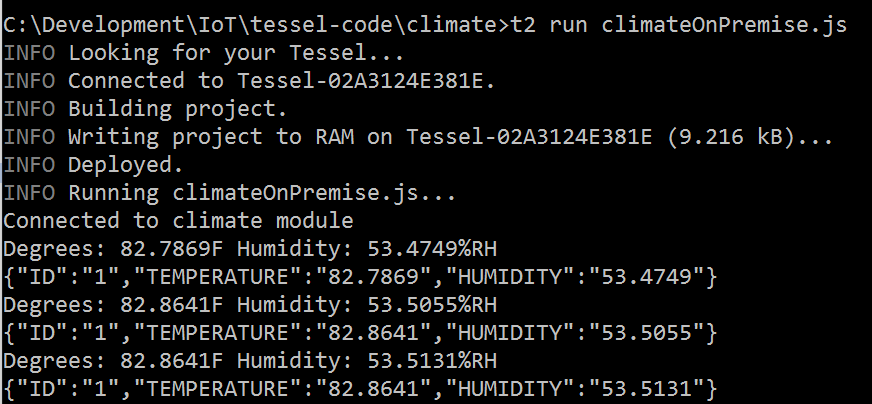
|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller",  "sap/m/MessageToast",  "sap/ui/model/json/JSONModel"  ], function (Controller, MessageToast, JSONModel) {  "use strict";  return Controller.extend("iot.controller.App", {  onInit : function () {  this.cModel = new sap.ui.model.json.JSONModel();  this.cModel.loadData("http://hd3.hana.ucc.uwm.edu:8003/GBI\_600/iot/iot.xsodata/DATA?$top=10&$format=json&$orderby=ID desc");  this.getView().setModel(this.cModel, "iot");    var oVizFrame = this.getView().byId("idVizFrameLine");  var oPopOver = this.getView().byId("idPopOver");    var oDataset = new sap.viz.ui5.data.FlattenedDataset({  dimensions : [ {  name : 'ID',  value : "{ID}"  } ],  measures : [  {  name : 'Temperature',  value : '{TEMPERATURE}'  }, {  name : 'Humidity',  value : '{HUMIDITY}'  }],  data : {  path : "/d/results"  }  });    oVizFrame.setVizProperties({  plotArea : {  isFixedDataPointSize : true,  categorySize : {  desktop : {  minValue : 100  }  },  dataLabel : {visible : true},    lineStyle: {  rules: [  {  dataContext: [  {ID: "\*"}  ],  properties: {  width: 6  }  }]  }  },  legend : {  title: {visible : false}  },    title: {  visible: true,  text: 'Line'  }  });  oVizFrame.setDataset(oDataset);  oVizFrame.setModel(this.cModel);  var feedPrimaryValues = new sap.viz.ui5.controls.common.feeds.FeedItem({  'uid' : "primaryValues",  'type' : "Measure",  'values' : ["Temperature", "Humidity"]  }), feedAxisLabels = new sap.viz.ui5.controls.common.feeds.FeedItem({  'uid' : "axisLabels",  'type' : "Dimension",  'values' : ["ID"]  });  oVizFrame.addFeed(feedPrimaryValues);  oVizFrame.addFeed(feedAxisLabels);  oPopOver.connect(oVizFrame.getVizUid());    var that = this;  this.refresh = true;  setInterval(function(){  if(that.refresh){  that.loadData();  }  }, 3000);  },    loadData: function(){  this.cModel.loadData('/destinations/iotbuiltin/NEO\_CT4FMF3FDW02NYG4UIPXV81BX.T\_IOT\_791B24BD6A2202BA3187?$format=json&$top=10&orderby=C\_TIMESTAMP%20desc');    },    doIt: function(){  this.refresh = !this.refresh;  },    degrees: function(temp){  return Math.round(temp) + "\u00b0";  },    humidity: function(hum){  return Math.round(hum)+ "\u0025";;  }  });  }); |

Listing 7

The added code will run the loadData() function every 3 seconds (3000 ms). The loadData() function simply fetches the data from the server again if the this.refresh variable is true. The doIt function toggles the this.refresh variable when the button is pressed. The variable this.refresh is initialized at the top of the onInit() function.

## Test the Climate App

If you implemented the Tessel climate module as shown in case IoT1Cxx – Setup a Tessel Device – On Premise, start the **climateOnPremise.js** program to begin loading data.



Run the UI application and carefully cup your hands around the Tessel climate module and breathe gently on it. Watch the temperature and humidity rise.

